**KWIC System Architecture with Abstract Data Types and Implicit Invocation**

The KWIC system uses an **Abstract Data Type (ADT) Architecture** and follows an **Implicit Invocation** design pattern. Below is a detailed breakdown of the system architecture, including components, connections, constraints, and the role of each class involved.

**1. Abstract Data Type (ADT) Architecture**

The architecture is designed around ADTs, where each component manages its data and operations. The system encapsulates the behavior of these components, which interact only through their exposed interfaces. This provides modularity.

* **LineStorage**: Abstracts the storage and retrieval of input lines and sorted shifts.
* **CircularShift**: Abstracts the generation of circular shifts for a line.
* **Alphabetizer**: Abstracts the sorting and merging of circular shifts.
* **InputModule**: Abstracts the input process, reading lines and passing them to storage.
* **Output**: Abstracts the output process, providing mechanisms to retrieve and print KWIC lines.

**2. Implicit Invocation**

Components communicate indirectly. Changes in one component trigger behavior in another. For example:

* **Master Control** implicitly invokes sorting in the **Alphabetizer** when a new line is processed. This coordination is handled without explicit method calls between components, allowing for loose coupling and flexibility.

**KWIC Components, Roles, and Connections**

1. **Master Class**
   * **Role**: Central coordinator of the KWIC system. Manages interactions between components and provides the interface for processing input lines.
   * **Components**:
     + **LineStorage**: Stores input lines and sorted circular shifts.
     + **InputModule**: Reads and stores input lines.
     + **CircularShift**: Generates circular shifts.
     + **Alphabetizer**: Sorts and merges circular shifts.
     + **Output**: Manages display of sorted KWIC lines.
   * **Key Method**:
     + **process\_line**: Orchestrates the process of reading lines, generating circular shifts, alphabetizing them, and updating the KWIC index.
2. **InputModule Class**
   * **Role**: Handles input by reading lines and passing them to **LineStorage**.
   * **Component**:
     + **lineStorage**: Instance of **LineStorage** where lines are stored.
   * **Key Method**:
     + **read**: Reads and stores the input line.
3. **LineStorage Class**
   * **Role**: Responsible for storing lines and their sorted circular shifts. Acts as a centralized data store.
   * **Attributes**:
     + **lines**: List of input lines.
     + **sorted\_shifts**: List of sorted circular shifts.
   * **Key Methods**:
     + **setline**: Adds a new line.
     + **getline**: Retrieves a specific line based on its index.
     + **word**: Returns the number of words in a line.
4. **CircularShift Class**
   * **Role**: Generates circular shifts for a given line.
   * **Attributes**:
     + **shifts**: List of circular shifts.
   * **Key Methods**:
     + **setup**: Initializes the process to generate circular shifts for a line.
     + **shiftWords**: Produces all possible circular shifts for the words in a line.
     + **CSLine**: Returns a specific circular shift based on its index.
5. **Alphabetizer Class**
   * **Role**: Sorts circular shifts in lexicographic order and merges them with existing sorted shifts in **LineStorage**.
   * **Components**:
     + **circularShift**: Instance of **CircularShift**, provides access to generated shifts.
     + **lineStorage**: Instance of **LineStorage** where sorted shifts are stored.
   * **Key Methods**:
     + **alpha**: Sorts and merges the circular shifts.
     + **ith**: Retrieves the i-th sorted circular shift.
6. **Output Class**
   * **Role**: Responsible for displaying sorted KWIC lines.
   * **Components**:
     + **alphabetizer**: Instance of **Alphabetizer** to retrieve sorted shifts.
   * **Key Methods**:
     + **print\_all\_KWIC**: Prints all sorted KWIC lines.
     + **print\_KWIC**: Prints the KWIC line at a specific index.

**KWIC Process Flow**

1. **Input (InputModule)**:
   * A line of text is passed to the system.
   * **InputModule** reads the line and sends it to **LineStorage**.
2. **Storage (LineStorage)**:
   * The line is stored in the **lines** list.
3. **Circular Shifting (CircularShift)**:
   * **CircularShift** generates all circular shifts of the input line. For example:
     + For "hello world", the shifts are:
       - "hello world"
       - "world hello"
4. **Sorting (Alphabetizer)**:
   * **Alphabetizer** sorts the shifts lexicographically.
   * The sorted shifts are merged into **LineStorage**'s **sorted\_shifts** list.
5. **Output (Output)**:
   * The **Output** class prints the sorted KWIC lines or a specific KWIC line.

**Connections Between Components**

* **Master**: Coordinates data flow across components.
* **InputModule → LineStorage**: Stores input lines.
* **LineStorage → CircularShift**: Provides lines for circular shifting.
* **CircularShift → Alphabetizer**: Passes shifts for sorting.
* **Alphabetizer → LineStorage**: Stores sorted shifts back.
* **Output → Alphabetizer**: Retrieves sorted shifts for display.

**Constraints for the KWIC System**

1. **Modularity**: Each component operates independently, following the ADT principle.
2. **Encapsulation**: Each component's internal data structure is hidden, exposing only essential methods.
3. **Separation of Concerns**: Components do not depend on each other's implementations, ensuring easier maintenance.
4. **Efficiency**: Sorting and merging in **Alphabetizer** must handle incremental updates without recomputing the entire index.
5. **Loose Coupling**: Components interact through implicit invocation, minimizing direct dependencies.

**Key Design Choices**

* **Separation of Concerns**: The modular design ensures that input, shifting, sorting, and output are managed by different components.
* **Incremental Processing**: Lines are processed individually, allowing continuous updates to the KWIC index.
* **Merging Sorted Data**: The system efficiently merges new shifts with existing sorted shifts in **LineStorage**.

This architecture balances flexibility, efficiency, and scalability, making it ideal for systems requiring modular design and dynamic data processing.